PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04L 29/06

A1

(11) International Publication Number: **WO 00/56028**

(43) International Publication Date: 21 September 2000 (21.09.00)

(21) International Application Number: PCT/CA00/00277

(22) International Filing Date: 15 March 2000 (15.03.00)

(30) Priority Data:

60/124,487	15 March 1999 (15.03.99)	US
09/515,092	29 February 2000 (29.02.00)	US

(71) Applicant: TEXAR SOFTWARE CORP. [CA/CA]; Suite 135, 1101 Prince of Wales Drive, Ottawa, Ontario K2C 3W7 (CA).

(72) Inventor: BACIC, Eugen; 56 Castlethorpe Crescent, Nepean, Ontario K2G 5R1 (CA).

(74) Agent: MITCHELL, Richard, J.; Marks & Clerk, P.O. Box 957, Station B, Ottawa, Ontario K1P 5S7 (CA).

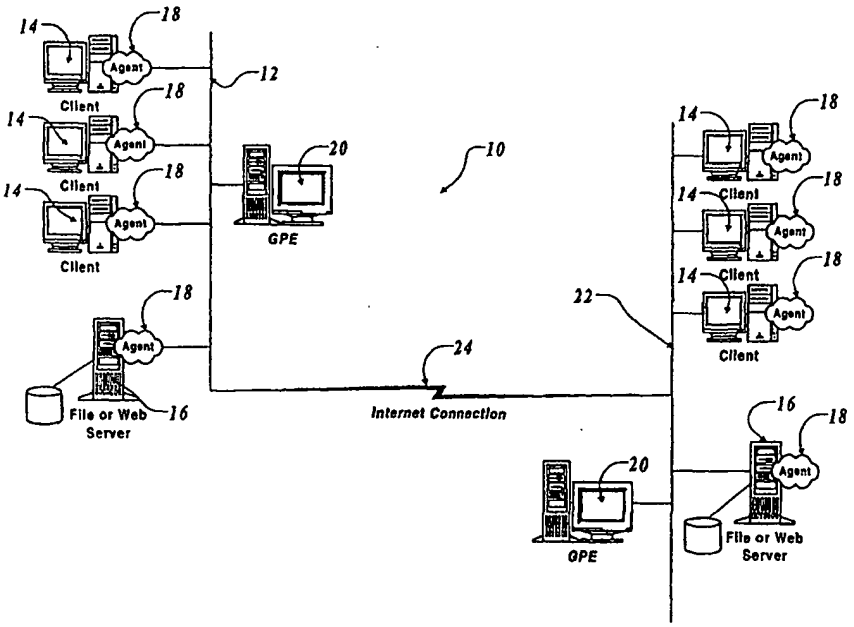
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: A SECURE NETWORK



(57) Abstract

A secure network includes a network server and a plurality of clients connected. A security server, typically physically separate from the network server, contains a database storing access rights for the network. A security agent at the network server controls access to the network server by communicating with the security server over the network in response to an access request from a client to determine access rights for the client.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

A Secure Network

Cross Reference to Related Application

This application claims the benefit under 35 USC 119(e) of US provisional application no. 60/124,487 filed on March 15, 1999.

Field of the Invention

This invention relates to the field of computer networks, and in particular to a secure network with sophisticated access controls.

Background of the Invention

As computer networks grow, security is becoming more of a concern with each passing day. Organizations view and relate to information differently and have differing requirements for the protection, dissemination, and modification of their information stores. Most organizations are moving towards heavily interconnected systems with links to the Internet. System architectures that were safe, due to limited accessibility, even a few short years ago are now being designed with wide access capabilities to meet the requirements of internal users, shareholders, customers, and clients.

For example, suppliers may wish to give third party customers limited access to their networks in order to facilitate design, ordering and accounting functions. Such third parties must not, however, have access to confidential corporate data, although in the case of co-operative design work, for example, there may be specific data files that the customer is authorized to access that would not normally be available to external organizations. This requires the ability to exercise highly sophisticated access control.

To meet security concerns, many organizations have opted for firewalls, virtual private networks (VPNs), and virus protection. These are commonly referred to as first generation security solutions. General access to host systems is provided based on the premise that once authenticated, users can be given full freedom to perform their duties.

Not only do first generation products protect only the perimeter, but they are islands of security, each performing a single task very well. They operate under their own control and their own rules; they do not play well with others, such as in the case of the third party customer example given above. These products have been defined and refined to

control external access to information and ensure that only legitimate users gain access to networks and their resources. Once a user is past these defenses, little, if any, security exists to protect valuable corporate information assets.

Windows NT and UNIX systems, for example, offer access controls, but these are often implemented incorrectly, or disabled altogether, due to previous bad experiences on the part of users or systems administrators. Moreover, they are sufficiently different to offer little help in integrating security.

First generation companies cannot obtain the trust required from their competitors to create integrated solutions. The fear of competitive advantage ensures that players will not engage in an integration effort. While some companies have emerged to provide a limited form of integration for audit, they do not address the other concerns that security officers have, nor is it clear that they will provide solutions in the foreseeable future.

Reliance on first generation products that protect the periphery of the information rather than the information itself is no longer sufficient. Security requirements are changing, placing new demands on security officers and system administrators. They must now fulfil six security demands:

- Integration among existing security products.
- A high water mark for overall network security.
- Controlled trust between systems even within an Intranet.
- Centralized and uniform controls.
- Compartmentalized and configurable information controls.
- Flexible and customer-oriented security rules.

First generation security products can be viewed as offering a wall of protection around information stores. Even though this wall may be sufficiently strong to thwart outside attack, it does not dissuade attack from the most common access point: the inside. These products are not designed to stop *authorized* users from accessing information. They are meant primarily to authenticate that a user is authorized to access the resources on the network. Once inside, few checks are performed, giving the user free rein.

Because of their distributed nature and the requirement to secure communications between machines, networked architectures are much more difficult to secure than single machines. This inherent difficulty is exacerbated by the fact that most networks do not have security systems and policies which were designed for the entire network. In most cases, security has been grown by combining preexisting security systems as individual machines are connected.

With advances in network computing, increased requirements to share information and processing power among physically separate locales, and worries about information sensitivity, existing security can no longer suffice. When two or more computer systems are linked, their security policies often clash and overall security actually diminishes. If policies have evolved over time, rather than being designed for the network, solving these problems becomes more difficult.

To deal with combined legacy architectures, encryption is used to at least guarantee that information gets safely to its destination. Encryption, however, is a coarse technique for protecting information from disclosure. With no standardized method of creating security policies, encryption has become the de facto method of protecting information network-wide, even though its use surrenders the fine-grained control that was available prior to the networking of the computers.

The need for a security infrastructure has been fueled by the desire to provide new solutions in the face of increasing incidents of unauthorized access to and manipulation of computer systems, data, and communications. Malicious misuse of computer systems can be classified into three groups based on the origins of the threat:

- **Outsider Threat** – This is an external individual or group attempting to breach the security of the system. Outsiders breach communications access controls but operate under the constraints of the communications protocols. This is the standard cracker attack. Outsider attacks are typically defended against by proper system administration and correctly designed and implemented access-control protocols and mechanisms, such as virtual private networks (VPNs) and firewalls.
- **Malicious Software** – This is a piece of malicious code introduced into the system. The attack takes place within the communications perimeter, but remains

bounded by the general access available to the operating system and the executing user. Malicious software may be introduced with or without a user's consent. The most common forms of this attack are the virus and Trojan horse.

- **Insider Threat** – Here, the perpetrator is an individual with authorized access to the system. An insider may have wide-reaching control of the system or its components. The perpetrator may opt to replace hardware or software, and may observe any communications channel. This attack occurs within the boundaries of the VPNs or firewalls as the perpetrator is an authenticated user. Insider threats commonly come from disgruntled or compromised employees.

A solid defense against insider threats can deter perpetrators by ensuring that they will get only a poor return on their investment. Such a defense can limit the damage done, minimize the information stolen or modified, and ensure that the perpetrator can be caught. In many cases, such a defense can stop most attacks, alerting authorities and ensuring that the threat is limited to system components which the insider generally has access to.

There is a need for a second generation security product that provides security across the network, security which is highly trustworthy, which is configurable to the needs of clients, which can be fully integrated with existing technology, and which is centralized for audit and administrative purposes.

Summary of the Invention

According to the present invention there is provided a secure network comprising a secure network comprising data storage means for storing protected data; a plurality of clients connected to said network; a security server containing a database storing access rights to said data separately from said data; and a security agent at said data storage means for controlling access thereto, said security agent communicating with said security server over said network in response to an access request from a client to determine the access rights of said client to said protected data.

The data storage means is typically located on a network server, but the invention is equally applicable to a peer-to-peer system, in which case the data storage means can be

distributed at the clients, or both. What is important is the fact that the security data is decoupled from the protected data and stored independently.

The security server is separate from the network server and it alone is responsible for controlling access to information on the network. This arrangement provides a security architecture that can offer a full range of integrated audit, authentication, access controls, and security policies. The invention offers a fully integrated solution that works with existing first generation solutions. The invention does not necessarily replace existing security solutions; it can complement and enhance them.

Due to its distributed nature, the invention can offer uniform and consistent enterprise-wide security across the corporate network infrastructure. As this solution is completely architecture-neutral, all corporate LANs, regardless of hosts, can be secured: Windows 95/98, Windows NT, Solaris, BSD, Linux, Macintosh, or any other.

The invention provides a set of layered security services addressing communications and data security problems in the emerging Internet and Intranet application space. The invention encourages interoperable, horizontal security solutions and offer the essential components of security capability to the industry at large.

The invention is applicable to a number of security applications including:

- Electronic commerce for business-to-business and home-to-business applications.
- Copyright-controlled content distribution of software, reference and entertainment materials.
- Metering of content and service.
- Secure storage of state and value.
- Securing business and personal activity on networks.
- Protecting information based upon client-specific business rules.
- Intelligent, security-aware information flow filtering.

These security requirements are addressed in several ways:

- Providing programmable, customer-centred security policies.
- Supporting application-specific policies by providing an extensibility mechanism that manages add-in modules for security policies and authentication systems.

- Supporting distinct user markets and product needs by providing an extensible security framework.
- Using a fully operational and flexible security interface that can accommodate a broad range of formats and protocols for mediation of information flow within an infrastructure.
- Supporting, utilizing, augmenting, and enhancing existing security products such as firewalls, VPNs, and certificate authorities.

The invention employs the generic policy engine (GPE) described in our copending application no. 60/124,487 filed on March 15, 1999, which allows programmers to create and generate a valid security policy from a high-level, verifiable semantic description of a security model. The invention utilizes an entity-based approach, defining entities for all security relevant components within its boundary. It provides one-for-one mapping between the entities to be protected in the calling application and the security information maintained by the system.

A functional language which is security aware and fully extensible defines the capabilities of the invention. The language allows the operation and development of security policies by means of the base language itself and by the security of the relevant library of functions. In this way it is possible to define any security policy that would be relevant to controlling information access or flow between a requesting entity and a piece of data.

Security relevant application programming interfaces (APIs) are provided so applications can manipulate security attributes, determine the proper information flow, call the security policy, examine audit logs, and perform regular maintenance on entities.

In another aspect the invention provides a method of controlling security in a network having a protected data storage means, comprising the steps of storing information access information in a security server; intercepting requests for information from said protected data storage means and passing them to said security server; and mediating said requests in said security server in accordance with security policies stored therein, said security server acting as a reference monitor, controlling the flow of information within the system.

Brief Description of the Drawings

The invention will now be described in more detail, by way of example, only with reference to the accompanying drawings, in which:-

Figure 1 is a diagram of a secure network in a business to business environment in accordance with the principles of the invention;

Figure 2 shows a secure LAN in more detail;

Figure 3 is a diagram of a secured LAN showing the administration functions;

Figure 4 is a high level schematic of the overall system architecture;

Figure 5 shows the high level architecture of the Generic Policy Engine;

Figure 6 shows the GPE SQL bridge software architecture; and

Figure 7 shows the discovery agent and related software components.

Description of the Preferred Embodiments

In Figure 1, the secure network comprises a pair of local area networks (LANs) 12, 22, typically associated with different businesses connected over a backbone 24. Each local area network 12, 22 has a plurality of clients 14, which can be running different operating systems, such as Windows NT, Unix, Windows 9x, etc and a local server 16.

Each local area network 12, 22 has a security server or generic policy engine 20, which includes a database containing access information pertaining to the clients or users on the network. The access controls can relate to the individual clients, or to specific users identified by password. The security server 20 is responsible for all access controls on its network.

Each LAN 12, 22 has a server 16, which typically would be UNIX or Windows NT based, for storing programs and data available to authorized users on the network.

In addition, each server 16 has an agent 18 at its access node to the network, which intercepts traffic to and from the network. Preferably, an agent 18 is also provided at each client, though that is strictly not necessary for operation of the system. When a client 14 makes a request for data, the request is intercepted by an agent 18, which then communicates with the GPE 20 to determine whether the request is authorized for that particular client/user based on information stored in the entity database 30. The agent 18

only provides access to the data when it receives an authorization message from the GPE 20. The agents communicate with the GPE using a messaging protocol, known as Plan Nein.

In addition to controlling access requests, the agents 18 at the clients can also perform identification and authentication functions (I & A) on the clients to verify the identity of users.

As shown in Figures 2 and 3, the system can work with a certificate authority, such as Entrust, which verifies the identity of users. The GPE 20 includes a monitor 26 and is connected to an entity database 30 via an SQL bridge 28, described in more detail below. The entity database 30 is populated by a discovery agent 32, which will be described in more detail below.

The GPE Server 20, which is a Java component of the system, provides security mediation between, and management of, entities within its domain of control. The entity in entity database 30 is the basic secured item. Each entity has an associated security policy written. Entity-related information is stored in a database for which a Java Database Connectivity (JDBC) driver has been written.

The GPE Server 20 manages sessions – connections via TCP/IP – to other devices, and clients 14 – users logged onto devices that have active sessions. It should also be noted that sessions have a single protocol associated with them, the Plan Nein protocol for sessions between agents 18 and the GPE server 20.

The core of the security system, shown in Figure 4, is built around the Generic Policy Engine (GPE) 20. The GPE 20 acts as a reference monitor, controlling the flow of information within the system. It enforces a security policy, permitting or denying transactions based on information about the security status of the requesting party, the object of the request, and the requested action. The GPE 20 performs the same generic mediation function in the same manner for all requests.

The core is made up of a set of security facilities and their associated application programming interfaces (APIs). These facilities provide six main functions:

- Audit
- Privilege

- Access Control
- Administration
- Identification and Authentication
- Security

These six facilities, along with the GPE itself, communicate through their APIs (Application Programming Interfaces) to a set of system agents.

The agents 18 provide a number of functions:

- Communicate between the core and network systems
- Collect and record information about the systems
- Logging security-related transactions to the GPE

The system security is based on an entity model. Everything on the system which is being secured – users, files, servers, etc. – is defined as an entity contained in database 30. Each entity is given a unique identifier, tagged with information denoting its access controls and privileges and security policy, and stored in a database.

When the GPE 20 receives a request to mediate a transaction, it examines the privileges of the requesting entity, compares them to the access controls of the requested object, and approves or disallows the transaction accordingly.

The core, shown in more detail in Figure 5, is designed in several layers, each of which may use the exported services of the ones below it. The layered design maximizes portability, allowing the system to port to many different operating systems.

The bottom layer of the system core is the native operating system of the machine running the security system, and its services. This is the one layer that will vary significantly between installations.

To ensure that system is fully portable between systems, the Java Virtual Machine (Java VM) will run on top of the native operating systems. All the software components belonging to this system that reside on the security server are in the form of Java byte code. This allows the software layers above base layers to be platform-independent. The

Java VM in effect acts as the security server operating system, providing full platform and operating system independence.

Both the operating system and the Java VM are third-party components used by the system. The first layer of system core technology is the language Interpreter, which facilitates the implementation and verification of security policies.

The Language Interpreter compiles code written in the Idyllic language into Java byte code, which is interpreted and executed by the Java VM.

The Security System is comprised of a number of components, shown in Figure 5. At the topmost level resides the GPE Shell. This shell provides a command line interface to allow operator control of the various features and attributes of the Generic Policy Engine. The GPE itself can communicate with various agents residing on the network via the Plan Nein protocol or the Management protocol, both of which sit atop the GPE Core.

The GPE Core is written primarily in Java with the security policy language, Idyllic, abstracted away from Java into a functional language of its own. The core GPE functionality and that offered by the security policies implemented within Idyllic can access the entity databases via the JDBC. This is done via two bridges: a GPE JDBC Bridge for the GPE proper and an Idyllic-based JDBC Bridge for the policy language.

The GPE can be viewed as a set of technologies that are responsible for the mediation of all requests for information access from the calling agents. This mediation is accommodated by the two custom, security-aware protocols: Plan Nein and Management.

The top layer of the system core consists of six security facilities. Each provides support for a particular set of security related operations.

Audit

The *Audit* facility allows the logging of all mediated transactions. The extent and detail of logging is controlled by the Administrator. Included in the audit facility are a minimum set of auditable events that cannot be turned off. These are hard coded into the audit facility, ensuring that the audit system cannot be accidentally or intentionally circumvented.

Privilege

The *Privileges* facility allows manipulation of the privilege set associated with entities.

Access Control

The *Access Control* facility provides replies to queries regarding the access rights of authenticated users to objects on the network.

Administration

The *Administration* facility provides basic services to the Administrator. These include the creation and control of entities, access rights, users and groups, and general maintenance of the system.

Identification and Authentication

The *Identification and Authentication* facility is responsible for logging users in and out and ensuring that they are not attempting to crack or spoof the system.

Security Policy

The *Security Policy* facility is tasked with defining and presenting the security policy chosen by the client. The implementation portion of the latter two security modules is supplied by the client or by third parties. These modules' APIs must conform to the set standard.

Agents

The agents 18 are the software components of system that run outside of the security server 20. They may include support utilities and embedded operating system components.

Designating all external components as agents and ensuring that they adhere to a consistent protocol when communicating with the security server 20 ensures that the possibility of abuse is minimized. It allows a single access methodology across all software components. The consistent definition of protocol between components also guarantees the ability to ensure the trustworthiness of not only the written agents, but also

all software written by third parties. Restricting the access methods to the core technology provides greater control over the traffic and interface, and thus greater trust, reliability, and security.

Security Policy

The system allows for programmable security policies. The security policy of the operating security system can reflect the actual operational mandate of the organization. Therefore, if the mandate is one of primarily a confidentiality concern, the security policy can be so defined. If the mandate is one with an integrity bent, then the security policy can be so defined.

The Security Policy agent provides an interface for the system Administer to define the security policy, including security-related entities.

Persistent Store

The *Persistent Store* management facility stores information and responds to queries about the entities in the system.

The primary purpose of the persistent store is to provide secure, long-term storage and retrieval of security-relevant data objects such as entities and security policies. The persistence of these objects is independent of the memory-based manipulations performed by the system. Persistent store modules may only be invoked by the GPE core.

The system's persistent store is based on an SQL database. The system includes an SQL Bridge agent which communicates between this database and the GPE.

Operating System Agents

The operating system agents 18 run on each machine within a network secured by the system. Their function is to intercept all file requests on the operating system, re-route them to security server, and allow or disallow them based on the result returned from the security server. File requests may originate from users logged directly to the machine running the OS, or they may come remotely from network connections. The agents are inserted in the OS such that no file request can circumvent the agent and result in a breach of security.

A different Operating System agent will be required for each type of operating system on the network.

Discovery Agent

The discovery agent is an agent that explores the system, discovering all objects, and reporting them to the system for cataloguing.

The discovery agent is run once on each machine belonging to a domain which is being secured by the system. This populates the database of entities 30. After the discovery agent has run successfully, the system controls all entities on the system and can make them available for secure transactions.

Due to the platform-dependent nature of the information being recorded, there is a separate implementation of the discovery agent for each operating system supported.

The discovery agent uses the Plan Nein protocol as a transport mechanism for communications with the security server 16 as will be described in more detail below

GPE

The GPE Server 20 will now be described in more detail. This consists of several packages. The high-level software architecture is shown in Figure 6. As can be seen in this figure, the GPE Server 20 consists of a GPE Shell , the GPE, the Plan Nein and GPE Management protocol layers, Idyllic and the Idyllic JDBC Bridge, and the GPE JDBC Bridge. Idyllic is the name for the customized language used in the implementation of this invention.

The GPE Shell has the responsibility of bringing up the GPE Server. It reads in the GPE initialization file (`gpeserver.ini`, by default), parsing its contents and initiating server threads for Plan Nein and GPE Management protocol connections.

The protocol server threads spawned by the shell have the responsibility for determination of acceptance or rejections of connections from specific IP addresses. Communication with the GPE Server occurs via sockets using either the Plan Nein protocol, through Port 333, or the GPE Management protocol through Port 334.

The Plan Nein protocol layer consists of two sub-layers. The lower layer is the protocol transport layer. This layer ensures correct framing of Plan Nein messages and delivers Plan Nein message objects to the upper Plan Nein Protocol Handler layer.

The GPE Management layer consists of two sub-layers. The lower layer is the protocol transport layer. This layer ensures correct framing of Management protocol messages and delivers these messages to the upper GPE Management Protocol Handler layer.

The GPE layer provides session and database management functionality along with application security. The GPE layer also manages the Idyllic environment. Separate Idyllic interpreters and environments are maintained for each client. This keeps binding namespaces for each client separate, making it impossible for one client to damage another's namespace through programmer error or malicious tampering. However, in order to have access to common Idyllic functionality, the GPE maintains a parent Idyllic interpreter for clients of each protocol. Client environments inherit bindings from their respective parents but, as before, may only extend their own environments at run time.

The GPE manages the Idyllic JDBC Bridge, synchronized with the GPE JDBC Bridge. When a database connection is established, the Idyllic JDBC Bridge bindings are added into the parent Idyllic interpreter's environment in order that clients may then begin to access the database. When the GPE closes the entity database, these same bindings are removed from the parent Idyllic's environment.

During the time that the GPE Server is in its started state, all access to the entity database is mediated by the GPE; the Plan Nein Protocol Handler passes all service requests to the GPE. A general principle in the architecture is that protocol handlers only see the GPE, nothing lower.

GPE SQL Bridge

Turning now to the SQL bridge 28, this is written is Java and has the responsibility for providing a single interface to the GPE 20 regardless of the underlying implementation of the physical database. The interface makes extensive use of the Java Database Connectivity (JDBC) facilities provided by Java Version 1.1. The JDBC design provides for the loading of drivers, written by database vendors, that hide the implementation details of the database from the application. Consequently, no changes to the interface

are required to access Microsoft SQL 7.0, IBM's DB2, Oracle's SQL, or other JDBC-compliant databases.

The interface relies upon the SQL-92 specification. This version of the standard was selected because all major vendors of SQL database products implement it.

The physical database is accessed by software of the JDBC layer, a vendor-specific driver loaded upon application initialization.

The Idyllic Interpreter accesses the database through the Idyllic JDBC Bridge layer. This layer has a number of important properties. Firstly, it is pluggable, i.e., it is only present when the GPE is ready to process client requests. Before the GPE is in placed in the started state, it is not possible for Idyllic to access the underlying database. Similarly, if the GPE moves from the started state to, say, the open state, this layer is removed. Secondly, the Idyllic JDBC Bridge layer has no knowledge of the underlying connection to the physical database; only the GPE JDBC Bridge layer has access to the Connection object. The Idyllic layer may only issue very specific read-only requests against the entity database. It is not possible to issue a general SQL query against the entity database.

These facilities have been provided to add security and robustness to the architecture. Security policies, potentially written by end users, are not considered trusted software elements and must be prevented from accidentally or maliciously damaging the entity database.

In the following two sections any words in *italics* represent method names.

The GPE JDBC Bridge

The GPE JDBC Bridge provides an abstraction layer between the GPE 20 and the physical database 30. The bridge provides the ability to manipulate the relational database records and tables in a way that is meaningful for the GPE. This layer provides the following database functions:

- Management
- Schema creation and destruction
- Query and update

- Security
- Lambda loader

The management interface provides the ability to *open* and *close* a database. They provide the ability to *create* and *destroy* a database respectively. The schema creation and destruction interfaces are defined in the GPEdatabaseCreatorInterface and GPEdatabaseDestroyerInterface interfaces respectively.

The query interface provides the ability to execute SQL queries against the underlying database and to manipulate the ResultSet[1] objects returned from these queries. It also provides application-specific query functionality to:

- Find specific entities
- Find all children of an entity
- Find the parent of an entity
- Find all owners of an entity, historical and current
- Find all access to, and modifications of, a given entity
- Find attributes associated with an entity, including security policy
- Update entity attributes and inter-entity relationships
- Load security policy lambda expressions for Idyllic interpreters

The security interface provides the ability to *grant* or *deny* access to specific sets of tables and table columns within a GPE database. This interface is used in conjunction with the creation of a database (primarily) in order to ensure that the GPE has read-write access to its own database.

The lambda loader interface provides a facility analogous to a Java ClassLoader. Essentially, it allows for a named lambda expression to be loaded into an Idyllic interpreter on demand. This interface may be implemented such that lambda expressions may be loaded from the GPE database. However, in future releases, on demand loading from parent GPEs may be supported.

---

[1] URL: http://java.sun.com/products/jdk/1.2/docs/api/java/sql/ResultSet.html

The Idyllic JDBC Bridge

The Idyllic JDBC Bridge is both pluggable and opaque with respect to the underlying JDBC Connection object. The principal responsibility of this layer is to facilitate access to persistent entity store with minimal user programming. This software layer implements a read-only interface for the GPE database and relies upon the functionality provided by the GPE JDBC bridge described in the previous section. The functions provided by the interface are:

- Find specific entities
- Find all children of an entity
- Find the parent of an entity
- Find all owners of an entity, historical and current
- Find attributes of entity and inter-entity relationships
- Find all access to, and modifications of, a given entity

The *add* and *remove* behaviour provides the ability for the Idyllic interpreter to access a database or have its access terminated. This is achieved by adding or removing bindings from the Idyllic interpreter's environment. ResultSet objects are **not** returned by methods defined in this class. Instead, specific columns from selected table rows are retrieved and List objects created from the results. These List objects are then manipulated in a familiar way by the Idyllic interpreter. *The Idyllic interpreter never sees low level SQL objects.*

The advantage of this design is that the underlying database may change completely without affecting the Idyllic Bridge software.

GPE Database Schema

The GPE database design consists of several linked tables, with the master table being the **entity** table. Each entity stored in the database has an associated 32 bit integer key, called the Entity Identifier, or *eid*. The *eid* is unique across all objects managed by the GPE. Therefore, our assumption is that no GPE will ever manage more than 2^{32} entities. A simplified view of the GPE database schema is shown in Figure 6.

In this section all words in **bold** represent database table names. Words in *italic* are field names.

The Entity table

The master table, **entity**, has 10 columns. The primary key is *eid*, an integer.

The *Name* field is a 256 character string that allows a user friendly description to be associated with the entity. The number 256 has been chosen as this represents the most common maximum length of filenames in an operating system.

The *EntityDataTableName* is a 64 character string whose value is the name of a table where entity-specific data is stored. For example, if the entity represents a user, *EntityDataTableName* might have the value, User. The **User** table might then have the columns: *name*, *last login time*, *authentication method* and other data. The *EntityDataTableName* can be thought of as providing the ability to subclass an entity by adding columns with entity-specific data. In the example shown in, *EntityDataTableName* would have the value EntityData and the fields in the extended entity would be *Version*, *Division*, *Domain*, *Machine*, *LocalObject* and *Identifier*.

The *IsActive* field is a bit field – a boolean – that stores whether the entity is currently able to participate in security policy decision making. If this field has the value zero, its associated security policy is considered inaccessible and security mediation requests return the result of the Global Default Security Policy for the entity concerned.

The *Expiry* field contains a date and time at which the entity requires re-confirmation of its status within the database. If the entity is not re-confirmed at this time, it is deleted from the database. *Expiry* may be thought of as implementing a lease mechanism not unlike that found in the Jini environment[2].

The *SecurityPolicyName* field is a 64 character string that points at the *name* column of the **SecurityPolicy** table. *SecurityPolicyName* is a foreign key, although we do not define it to be so as we do not automatically delete the entity when the referenced named security policy is deleted (as would normally occur).

The *Owner* field is a 32-bit integer which stores the *eid* of another entity which is the owner of the entity identified in *Name*.

---

[2] URL: http://www.sun.com/jini/

The *LastModified* field is a date and time field. The value stored here represents the point in time at which the entity was last changed in some way.

The *LastAccess* field is a date and time field. The value stored here represents the point in time at which the entity was last accessed.

The *isPrivate* field is a bit field.

The SecurityPolicy table

The **SecurityPolicy** table has two columns, *name* and *lambda*.

The *name* field is a 64-character string that represents the unique name of a security policy. It is the primary key for the table. This field is referenced by the *SecurityPolicyName* field of the **entity** table.

The *lambda* field is an 8000-byte string that stores the S-expression representing the security policy. It is this table that is accessed by the LambdaLoader class. While 8000 bytes may not seem a large string when one considers the complexity of potential security policies, it should be noted that one policy may reference another, thereby making it possible to build elaborate policies from small building blocks, much as is done with standard software applications or components.

The RelatedEntity table

The title of this section is a misnomer as there is no actual table called **RelatedEntity**. There are instead 13 tables, with the names as shown in Figure 6. All these tables have the same columns: *eid*, *eid2*, and *allow*.

The *eid* and *eid2* fields contain 32-bit integer values that represent entity identifiers. Both columns are foreign keys and are related to the primary key of the **entity** table.

The *allow* field is a bit field that represents the not operator for the relationship. More clearly stated, if the table name were **read,** the tuple <*eid*, *eid2*, *allow*> were <98993, 100456, 0> would be taken to mean, "entity 98993 may read entity 100456." In other words, the *allow* column represents a relationship which does apply.

Table 1 explains the meaning of the various fields.

Table 1

Name	Meaning
Child	eid is a child of eid2
Parent	eid is a parent of eid2
Group	eid is a member of group eid2
Level	eid is a member of level eid2
Categories	eid is a member of category eid2
Read	eid has read access to eid2
Write	eid has write access to eid2
Execute	eid has execute access to eid2
Copy	eid can copy eid2
ReferencedBy	eid is referenced by eid2. This is not used in release 1.0 and is used in conjunction with the **ExternalEntity** table described in the next section.

The ExternalEntity table

This table is for use in GPE Servers that are linked in hierarchical relationships. When such a situation arises, it is necessary to have references to externally-stored entities. This is the responsibility of the **ExternalEntity** table.

This table has three columns. The *localEID* field is an integer entity identifier that represents the local view of an externally-stored entity. The value stored in this field may appear in the *eid* or *eid2* fields of **RelatedEntity** tables. A value used in *localEID* may not be used in the *eid* field of the **entity** table and vice versa.

The *URL* field is a 256 character string that stores the access protocol and location of the externally-stored entity. The access protocol is included here in order to allow for the definition of alternate access mechanisms that may need to traverse firewalls (for example).

The *remoteEID* column is an integer entity identifier that represents the remote view of the externally-stored entity. By having local and remote views linked through this table we do not need globally unique entity identifiers.

Discovery Agent Specification

The discovery agent 32 (Figure 2) is responsible for populating the entity database 30 as noted above. The discovery agent, shown in more detail in Figure 7, is an agent that runs in privileged or supervisor mode on a machine. The discovery agent explores the machine, discovers all objects, and reports them to the Generic Policy Engine (GPE) 20 for cataloguing.

The discovery agent 32 is run once on each machine belonging to a domain under the aegis of the security server 16. This initial step must be performed by the *security administrator* user before the system can accede to requests for data. This step populates the database 30 of controlled objects, known as *entities*. After the discovery agent 32 has run successfully, the security system controls all entities on the machine and can therefore make them available for secure transactions.

Due to the platform-dependent nature of the information being recorded, there will be a separate implementation of the discovery agent for each operating system supported.

The discovery agent uses the Plan Nein protocol as a transport mechanism for issuing transaction requests to the security server 16.

Basic Operations

Objects catalogued by the discovery agent include user logins, user groups, logical disk drives and their files, and other logical and physical devices.

When the discovery agent encounters an object to be catalogued, it will issue a request to the GPE for the creation of a new entity for the object. If the transaction is successful, the GPE will respond with a newly-generated unique entity identifier, or *eid*. This can later be used as a reference to the entity.

Once the create transaction has been acknowledged, the discovery agent issues update transactions that include the eid, the name of the entity attribute being updated, and the new value of that attribute.

If the object in question is nested, such as a file directory, the discovery agent opens it and processes all its elements recursively.

At the GPE end, each **create** or **update** transaction is validated. If a transaction is accepted, the GPE invokes its Persistent Store API to perform the associated database

transaction on the specified entity. If the transaction is not accepted then a rejection notification is returned to the discovery agent.

Discovery Agent Operation

The discovery agent progresses through seven phases of operation:

1. Phase 0 – Launch
2. Phase 1 – Initialization
3. Phase 2 – Cataloguing users
4. Phase 3 – Cataloguing user groups
5. Phase 4 – Cataloguing files
6. Phase 5 – Cataloguing devices
7. Phase 6 – Shutdown

On some platforms it is possible to omit Phase 3. The phases following Phase 1 are highly platform-dependent.

Phase 0: Launch

When a new machine is to be added to the control domain, the security administrator issues the appropriate command from the *Admin Console* shown in Fig. 5. This causes a message to be sent directly to the Administration API, directing it to start the discovery agent.

The Administration facility then repackages the start command, including the eid machine prefix for the machine to be catalogued, and calls the GPE API. The GPE in turn calls out to the appropriate machine via the *Dispatcher*, causing the discovery agent to begin execution.

Phase 1: Initialization

The discovery agent authenticates itself to the system and awaits permission to proceed. The discovery agent then initializes its internal cache. The machine itself is catalogued by sending a **create** transaction to the GPE with the name field set to the machine's name.

Phase 2: Cataloguing Users

In this phase the list of user logins is opened and traversed. On some operating systems, such as UNIX, these are read from a disk file. On other systems they are read from a system repository (e.g., the Windows Registry) or using a system API call.

For each user encountered three actions occur:

1. A **create** transaction is issued.
2. Entity fields are filled in using **update** transactions. These include the entity name (*uid*), the list of access rights associated with the user (if applicable), and the last modified and last accessed dates (from the last login date).
3. The eid/uid pair are cached for use in Phases 3 and 4.

Phase 3: Cataloguing User Groups

In this phase, the list of user groups (sometimes called workgroups) is opened and traversed. On some systems these do not exist, in which case this phase is skipped. The group information is obtained either directly from a file (as on UNIX, for example) or via system API calls. This process is similar to that of Phase 2.

For each group discovered four actions occur:

1. A **create** transaction is issued.
2. Entity fields are populated using **update** transactions. These include the entity name (from the human-readable group name, or, if present, the numeric group id, or *gid*), the list of access rights associated with the group (if applicable), and the last modified and last accessed dates (from the last login date, if applicable).
3. The eid/gid pair are cached for use in Phase 4.
4. For each group, each constituent user's eid is looked up in the local cache. An **update** is then issued for the user's eid to add the group's eid to its Groups privilege list.

Phase 4: Cataloguing Files

In this phase, the machine's entire file tree is catalogued. Each permanently mounted logical disk drive present on the system is catalogued. Starting at the root, each file tree is walked and every directory and file encountered is catalogued. No assumption is made as to whether the directory entries are ordered lexicographically or by file type (e.g., subdirectories first, followed by regular files.)

Scratch files are catalogued, as these are frequently used for hiding Trojan horses or covert channels. The security administrator can remove them from the catalogue at a later date if this is deemed appropriate. If the operating system includes devices in file trees, these are skipped (they are handled in Phase 5). Soft links to files residing on other machines are entirely skipped.

For each directory entry, two or three actions are performed:

1. A **create** transaction is issued.
2. Entity fields are populated using **update** transactions. These include the Entity name (from the absolute path of the file), the owner (from the eid corresponding to the uid of the file owner), the list of access rights (from the assigned file access attributes), and the last modified and last accessed dates (from same). Some file systems do not store a last accessed date; in such cases the last modified date is used.
3. If the directory entry is a file directory, its subdirectories are catalogued recursively.

Phase 5: Cataloguing Devices

Devices catalogued in this phase include logical devices, physical devices, I/O ports, and network connections. Some operating systems group these in special subdirectories or make them appear as files having special file attributes. Other operating systems enumerate them in a system repository, such as the Windows Registry. Human-readable names are utilized for all devices and ownership is granted to the host machine. Attributes such as last accessed and last modified are set to some logical value, typically the current time, but all these can be adjusted by the administrator later. The goal is to ensure

unauthorized use of these devices is stopped ensuring sensitive information does not leave the machine.

Phase 6: Shutdown

The final phase of the discovery agent operation consists of cleaning up. Dynamically allocated data objects are destroyed and memory and other resources are released. Finally, the agent ends its session and terminates.

In the event that a fatal error occurred in one of the previous phases, Phase 6 will roll back all issued transactions. This means that all entities created for objects on the local machine are removed.

The GPE performs transaction checkpointing. Consequently, the discovery agent can rescind a block of transactions easily, should an irrecoverable error occur. The performance benefit of this scheme is that it permits the rollback of many sequential transactions without incurring a large volume of network traffic.

At the beginning of a transaction block, the discovery agent 32 synchronizes with the GPE 20 by issuing a begin transaction. The discovery agent then performs all the transactions required to populate the system's Persistent Store with entities representing objects discovered on the machine being scanned. If Phase 6 is reached and the scan has proceeded successfully, then the discovery agent issues a commit transaction to the GPE. This directs the GPE to commit all transactions in the current block. In the case of fatal error, a rollback transaction would be issued, canceling all transactions in the block.

Homogenous Security in an Heterogeneous Environment

Compatibility with existing secure and non-secure software lets the system provide security enhancement across the enterprise or within a well-defined domain, such as a single department. It offers a single point of reference for all transactions requiring secure mediation. A request for information, be it from the web server, the FTP server, a file server, or an individual desktop, can be mediated consistently by the security server. All systems can offer identical levels of security which have been defined to meet the customer's needs, from the C2 level security common in commercial UNIX systems to the A1 security demanded by the military.

As a fully distributed solution, each security system can communicate with its neighbors to ensure that corporate security is maintained. In addition, it does not need to be deployed company-wide but can be used only within the most sensitive areas, allowing for a controlled rollout. As more products are brought online in an organization, the security of the information increases. When the security operates across the enterprise, information is protected to the maximum extent possible.

Each element of information can be protected according to where and how it was created. This allows information to carry its own security policy between domains in an organization. As information is transmitted between authorized parties, the security policy defining how that information is to be handled is also transmitted, even if the recipient's security policy differs substantially from that of the sender.

One example of an agent 18, suitable for a Unix machine, will now be described. The Unix agent's primary purpose is to mediate access to various Unix system resources by users and processes. Users and processes are known as *actors*. Actors perform *actions* with or on resources. All running processes are considered actors because they make use of the resources on a system and require interaction with the operating system. Users are represented by the interactive shells they use and the programs that the shell initiates. The shells interact with the operating system and make the requests for resources. There are a variety of possible requests that actors can make of the operating system involving many different types of resources. These resources that are acted upon can be files; communication channels, such as network communication or interprocess communication (IPC); shared memory; device access, such as tape drives, modems, sound systems, CDROM drives, floppy drives, or optical drives; and kernel services. These resources are known as *elements*.

OS agent

The Unix agent 18 is responsible for intercepting any kind of request for access to elements that actors ask of the operating system. In Unix, this task is normally handled by the kernel. The Unix agent intercepts these calls, gathers some information about the request, sends a query to the GPE, and waits for a reply. If the reply is to allow the request, the agent will pass the request to the operating system kernel, which will process

it as it would have in the first place. If the reply is to deny the request, the agent will not pass the request on to the operating system, returning a failure code to the requesting actor.

The Unix agent is intended to monitor a number of elements of the operating system. The monitoring of elements is most likely to be realized through the monitoring of system calls. This might not apply to all elements or even to all Unix variants, but should offer sufficient functionality to accomplish the principle goals of the agent.

The most important of these are file access and network access. The file element and the network elements will be the focus of most monitoring activity. In most variants of Unix operating systems, these two elements can be covered by monitoring a small group of system calls.

Traditional Unix systems use monolithic kernels as their core supervisor program. This makes it hard to distribute applications such as a Unix agent, which effectively needs to be part of the kernel. To distribute a kernel-type product for different platforms, the source code, or at least a library together with some source code, need to be distributed to customers. The setup time for this becomes an issue and some proprietary source becomes visible.

Modern Unix vendors now distribute operating systems with runtime loadable module support. A design goal of the agent is that it be developed as a runtime module for as many operating systems and architectures as possible, where such facilities exist. This makes distribution much easier, with only operating system versions being the sole reason for recompilation. There is almost no risk of giving away any proprietary secrets.

The agent must be made as transparent as possible to the operating system. This means that users must not, where possible, know or suspect that an agent is monitoring the system.

The agent must not, where possible, conflict with or hinder in any way the normal operation of the system on which it resides. There should be no loss of functionality to the system or the users of the system.

The agent must be able to use a communications channel directly to an outside source. This communication channel should be a widely used transport/network protocol such as

TCP/IP. The use of sockets is greatly recommended for as much compatibility as possible.

In the event that it is not possible to communicate directly to an outside source from the agent, a suitable intermediate method should be employed. The method must be secure against unauthorized use, spoofing or abuse for covert communications. The method used must not leave any tell tale signs of its actions or leave any information that originated with the agent, the centralized GPE or the communication session between these two.

The agent will communicate with the GPE using a proprietary application layer protocol. The protocol stack should be designed and coded in such a way as to be easily replaced with another protocol stack. This enables the agent to be more easily integrated into other systems where the need to use another protocol is required. The protocol stack is integrated into the agent, necessitating a recompilation of the source code in the event a different stack is required.

The agent should be designed and coded so that extra functionality can easily be added with a recompilation. This will not be possible in every case, but good design practices shall take precedence over ease of coding. An example of added functionality is an agent that presently monitors file access becomes one that monitors file access and sockets. The basic architecture of the agent should allow for adding or removing such functionality without completely rewriting the module.

An example of such an agent is designed as a kernel module for a Unix machine, and in particular is designed for use with FreeBSD 3.1.

The system's call interface has to be monitored and system calls intercepted. In order to do this, the system call table in the kernel will has to be modified by replacing the existing calls with special calls for the security system and saving the old calls for the execution of the original request, should the GPE allow it. To do this, we make use of the KLD module loading routine. The loading routine is actually a kernel function that is called from user land via a generic system call. All work done inside the kernel will never be interrupted since the kernel is never preempted; thus the replacement of the system calls is an atomic action.

The system call table is an array of structures. The structure is called *sysent* and is defined in `/usr/include/sys/sysent.h`. The array is initialized in file `/usr/src/sys/kern/init_sysent.c`.

The system structure is composed of two entries:

```
struct sysent {                 /* system call table */
     int   sy_narg;         /* number of arguments */
     sy_call_t *sy_call;  /* implementing function */
};
```

It is very important to note that the index of the sysent array corresponds to the index of the system call *name* array, *syscallnames[]*.

To clarify, the system call *open* in the syscallnames array is at index 5. Therefore, the corresponding kernel function for the open system call can be found in the sysent array at index 5. Conversely, you can use the system include file `/usr/include/sys/syscall.h` to index into the sysent array. The structure found in the sysent array holds two members: the first is the number of arguments to the system call, and the second is a pointer to the implementing function.

The KLD Module Interface makes use of the system linker for loading files into kernel space and making the relevant table entries for the linked list of modules.

The file resides in `/usr/src/i386/kern` and is called `kern_linker.c`.

In accordance with the invention, the security system is the sole mediator for all information requests. By brokering all requests, trust between systems, networks, and users can be defined. This provides assurance that information will remain where it belongs unless an authorized individual attempts access. Blanket trust of similar systems is not provided, and the general rule of operation for the system is to deny access unless it is explicitly allowed by the security policy.

As the security system is the sole mediator, all controls are centralized. These controls are uniform in form and appearance. The centralized administrative facilities allow security officers and administrators to access restricted areas to fine-tune security and

examine critical security information, such as audit logs. To maximize functionality, the administrative utilities are written in Java™ to provide a uniform look-and-feel, regardless of the platform a site uses for administrative purposes.

For information to be properly protected, it must be compartmentalized. Most security products today compartmentalize at the file or directory level. The system according to the present invention has no such restrictions. As it mediates information access based on an object-oriented approach, it does not matter whether information is compartmentalized to the directory level, the file level, the page level, the paragraph level, or the byte level. The compartment size is definable by the customer.

The smaller the information object being protected, the less efficient the mediation. If every byte must be brokered by the security system, performance will be seriously hindered. Under normal use the product will operate based on normal-sized pieces of information; files and pages for most institutions, and files and paragraphs for the military.

Unlike other security products on the market, the security system provides for true security policy customization. It can mediate regardless of the security policy a specific piece of information may require. If information migrates from one domain with an integrity policy to another with a disclosure or confidentiality policy, the system can understand the security requirements of the policies and act accordingly. Security policies are not restricted to those provided by the system supplier; customers can use powerful administrative functions to customize security policies that accurately reflect their own operational practices, something no other product can offer.

Security without performance and transparency is worthless. The present invention is fast and transparent to end-users, as long as they are performing their duties and not attempting actions which contravene the security policy. If users are performing their duties, they will not be aware of the actions of the underlying security system, ensuring that valid access attempts are quickly permitted and invalid ones quickly disallowed.

We claim:

1. A secure network comprising:

data storage means for storing protected data;

a plurality of clients connected to said network;

a security server containing a database storing access rights to said data separately from said data; and

a security agent at said data storage means for controlling access thereto, said security agent communicating with said security server over said network in response to an access request from a client to determine the access rights of said client to said protected data.

2. A secure network as claimed in claim 1, wherein said data storage means is located on a network server.

3. A secure network as claimed in claim 1, wherein said security agent, in response to an access request, exchanges control messages with said security server to determine said access rights.

4. A secure network as claimed in claim 3, further comprising a security agent at each client for intercepting access requests to said data storage means and obtaining authorization from said security server in response to said access requests.

5. A secure network as claimed in claim 2, wherein said security server is physically separate from said network server.

6. A secure network as claimed in claim 1, wherein said security server contains a database based on an entity model, each item on the network to be secured being defined as an entity, each entity describing the security attributes of an item to be protected.

7. A secure network as claimed in claim 6, wherein each entity has a unique identifier, tagged with information denoting its access controls, privileges and security policy.

8. A secure network as claimed in claim 6, wherein said database is accessed via an SQL bridge.

9. A secure server as claimed in claim 1, wherein the security server has a layered core structure, each of which uses the services of the underlying layer.

10. A secure network as claimed in claim 9, wherein bottom layer of said core is the native operating system of a physical machine, and said virtual machine is implemented above said bottom layer.

11. A secure network as claimed in claim 10, wherein a security language is implemented above said virtual machine to provide APIs for programmers to customize a security policy.

12. A secure network as claimed in claim 11, wherein said virtual machine is implemented in JAVA and a language interpreter layer is located between said JAVA-implemented virtual machine and said security language layer.

13. A secure network as claimed in claim 1, wherein said security server is connected to an authentication authority to authenticate the identity of users of the system.

14. A secure network as claimed in claim 1, wherein said agent intercepts all file requests on the operating system of said network server, re-routes them to security server, and allows or disallows them based on the result returned from the security server.

15. A secure network as claimed in claim 1, wherein a said agent runs on each machine within the network secured by the security server.

16 A secure network as claimed in claim 15, further comprising a discovery agent that explores a network device, discovering all objects, and reporting them to the security server for cataloguing in the database.

17. A method of controlling security in a network having a protected data storage means, comprising the steps of:

storing information access information in a security server;

intercepting requests from information from said protected data storage means and passing them to said security server; and

mediating said requests in said security server in accordance with security policies stored therein, said security server acting as a reference monitor, controlling the flow of information within the system.

18. A method as claimed in claim 17, wherein said server stores said security policies in a database based an entity model, each entity storing the security attributes of an item to be protected.

19. A method as claimed in claim 18, wherein each entity has a unique identifier, tagged with information denoting its access controls, privileges and security policies.

20. A method as claimed in claim 19, wherein when the security server receives a request to mediate a transaction, said security server examines the privileges of the requesting entity, compares them to the access controls of the requested object, and approves or disallows the transaction according to the information stored in the entity database.

21. A method as claimed in claim 17, wherein the security server works with an authentication authority to authenticate the identity of users.

22. A method as claimed in claim 18, wherein the security server communicates with said database via an SQL bridge.

23. A method as claimed in claim 17, wherein said requests are intercepted by an agent which communicates with said security server to mediate said requests.

24. A method as claimed in claim 23, wherein a said agent is located at said security server at the operating system level to intercept said requests.

25. A method as claimed in claim 23, wherein a said agent is run on each machine connected to the network to be secured.

26. A method as claimed in claim 18, wherein a discovery agent explores a network device, discovering all objects, and reports them to the security server for cataloguing.

27. A method as claimed in claim 26, wherein the discovery agent is run once on each machine belonging to a domain which is being secured by the security server to populate the entity database.

28. A method as claimed in claim 17, wherein said protected data storage means is located on a network server.

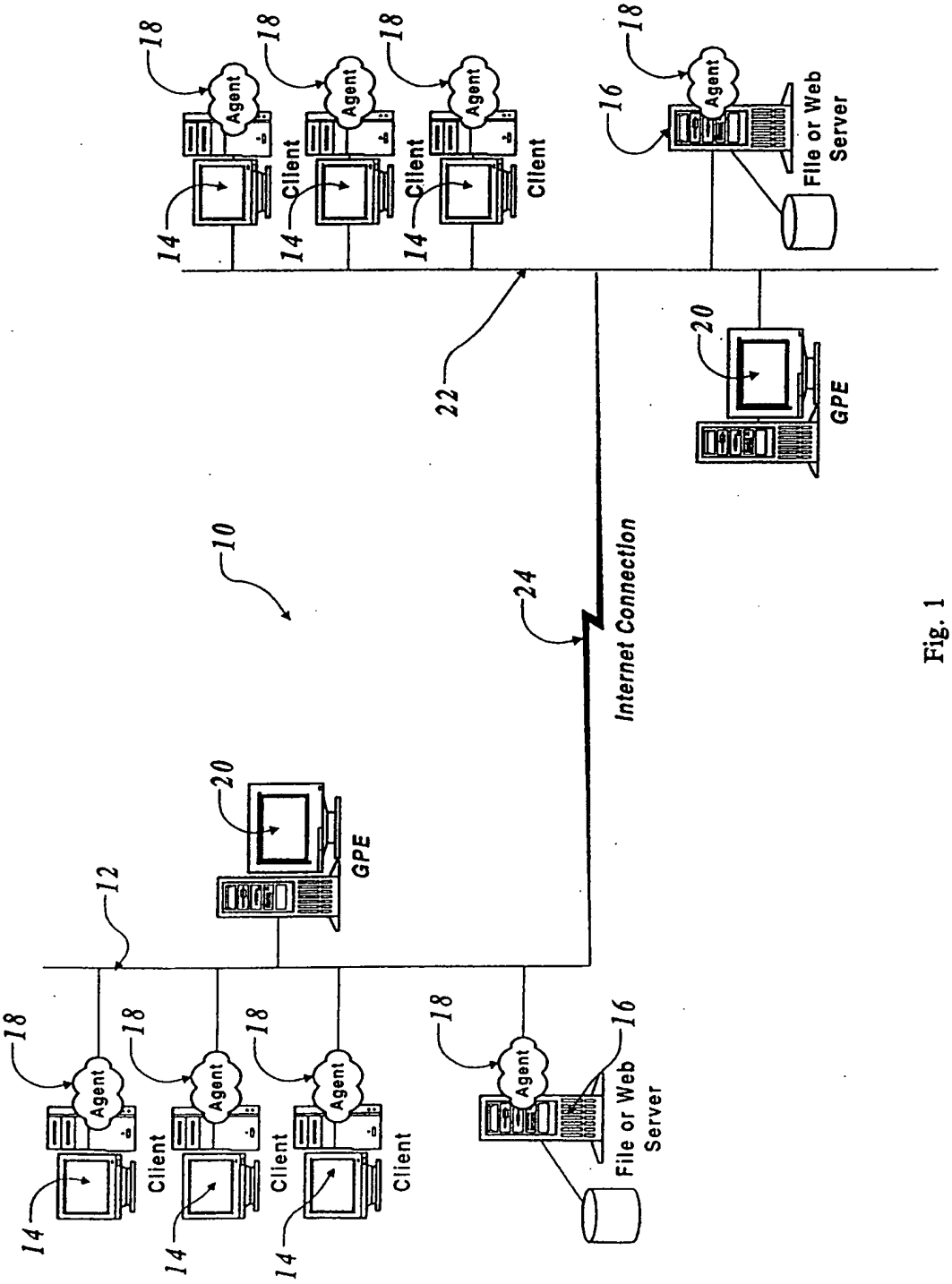

10
12
14
16
18
20
22
24
Agent
Client
File or Web Server
GPE
Internet Connection

Fig. 1

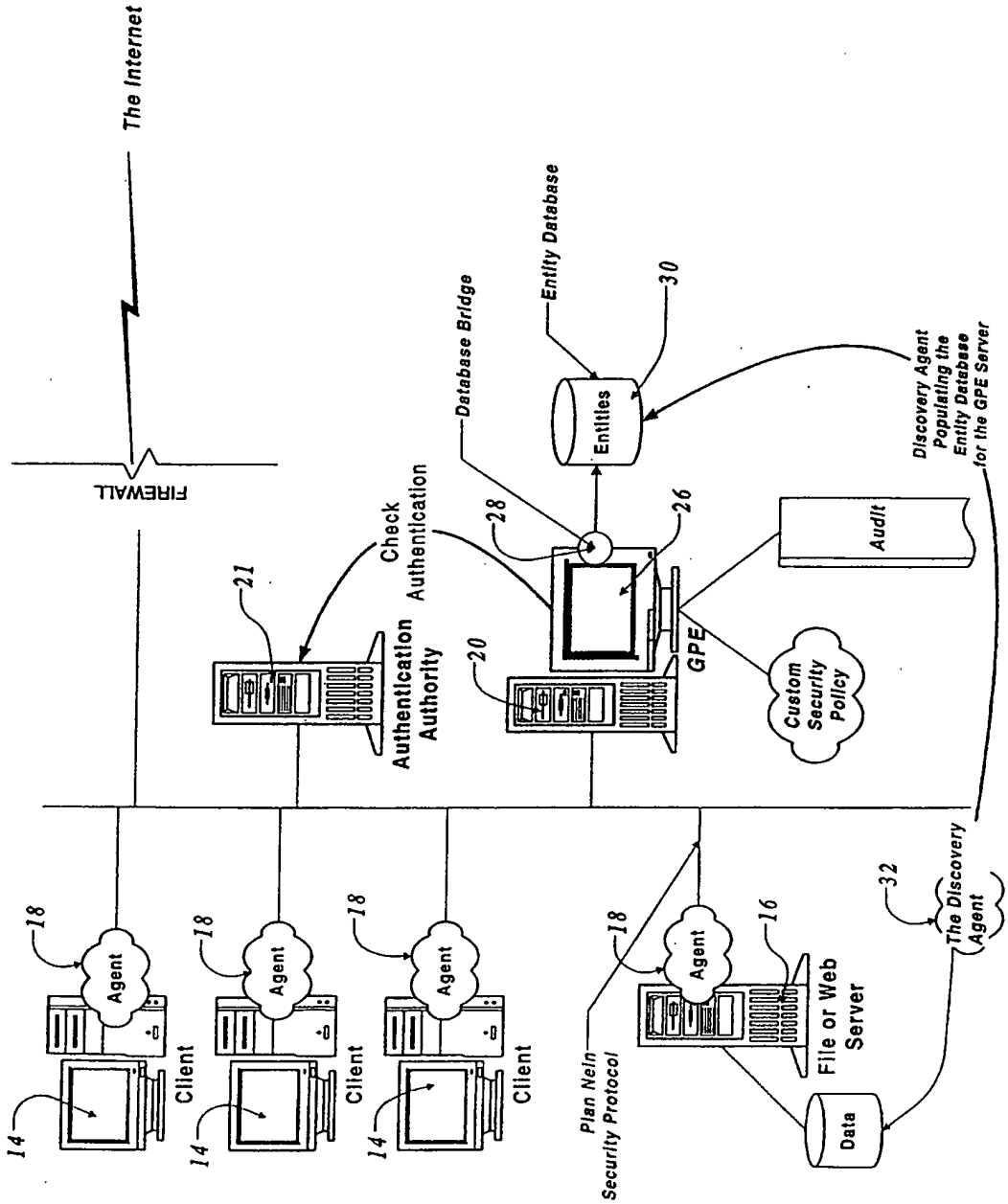

The Internet
FIREWALL
Check Authentication
Database Bridge
Entity Database
30
Entities
Discovery Agent Populating the Entity Database for the GPE Server
28
26
21
Audit
Authentication Authority
20
GPE
Custom Security Policy
18
Agent
Client
14
Plan Nein Security Protocol
16
File or Web Server
Data
32
The Discovery Agent

Fig. 2

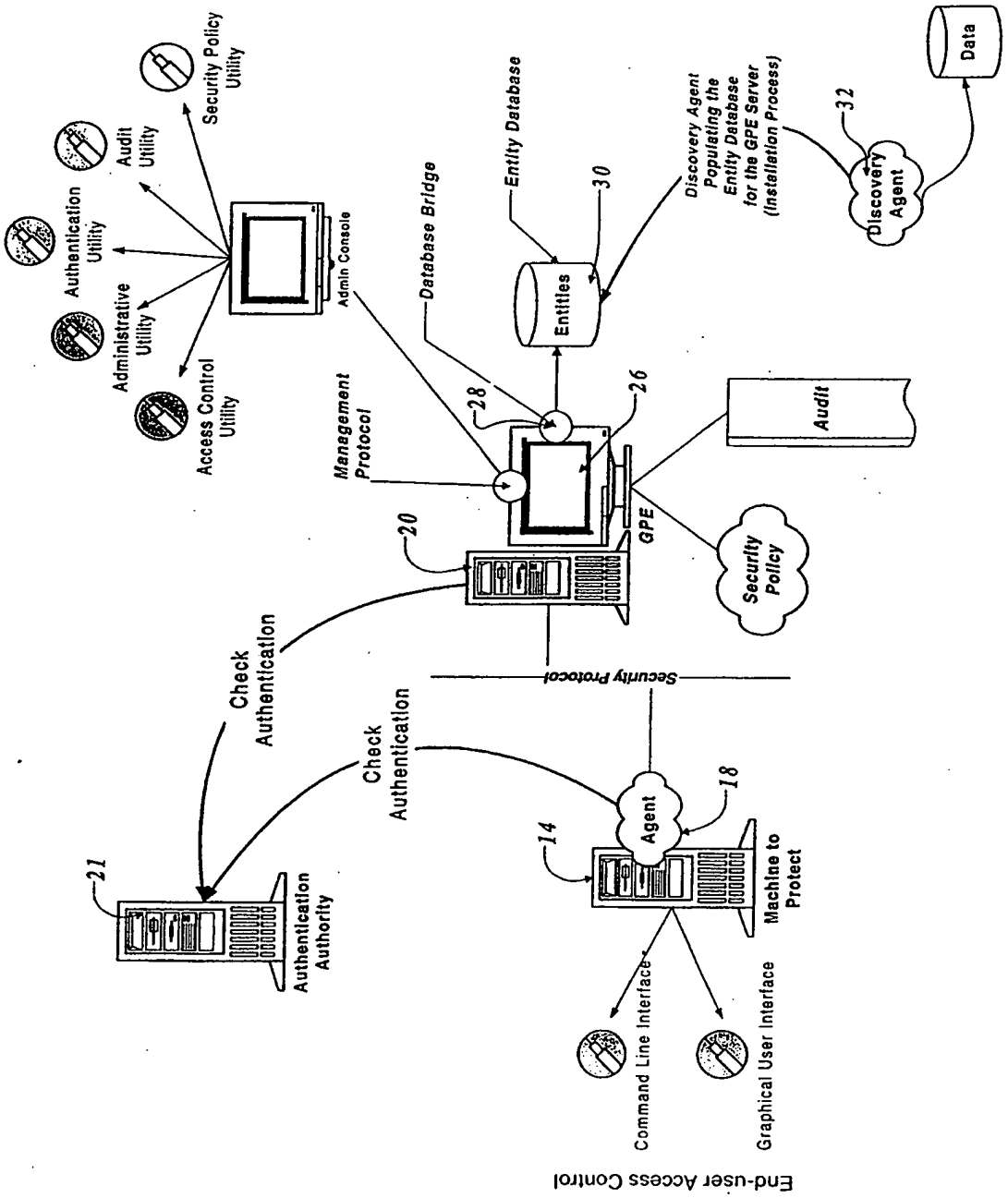

Security Policy Utility
Audit Utility
Authentication Utility
Administrative Utility
Access Control Utility
Admin Console
Database Bridge
Entity Database
30
Entities
Discovery Agent Populating the Entity Database for the GPE Server (installation Process)
32
Discovery Agent
Data
Management Protocol
28
26
Audit
GPE
20
Security Policy
Check Authentication
Check Authentication
Security Protocol
Agent
18
14
Machine to Protect
21
Authentication Authority
Command Line Interface
Graphical User Interface
End-user Access Control

Fig. 3

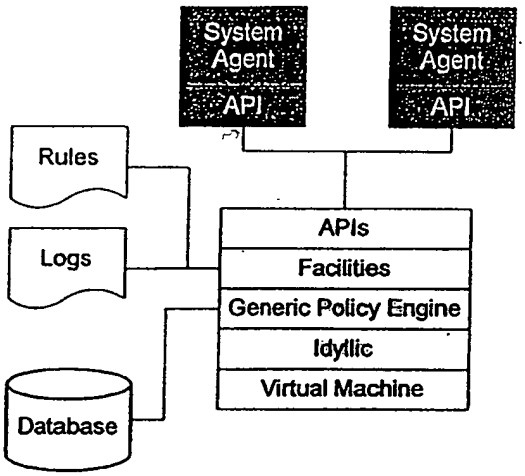

System Agent
API
System Agent
API
Rules
Logs
Database
APIs
Facilities
Generic Policy Engine
Idyllic
Virtual Machine

Fig. 4

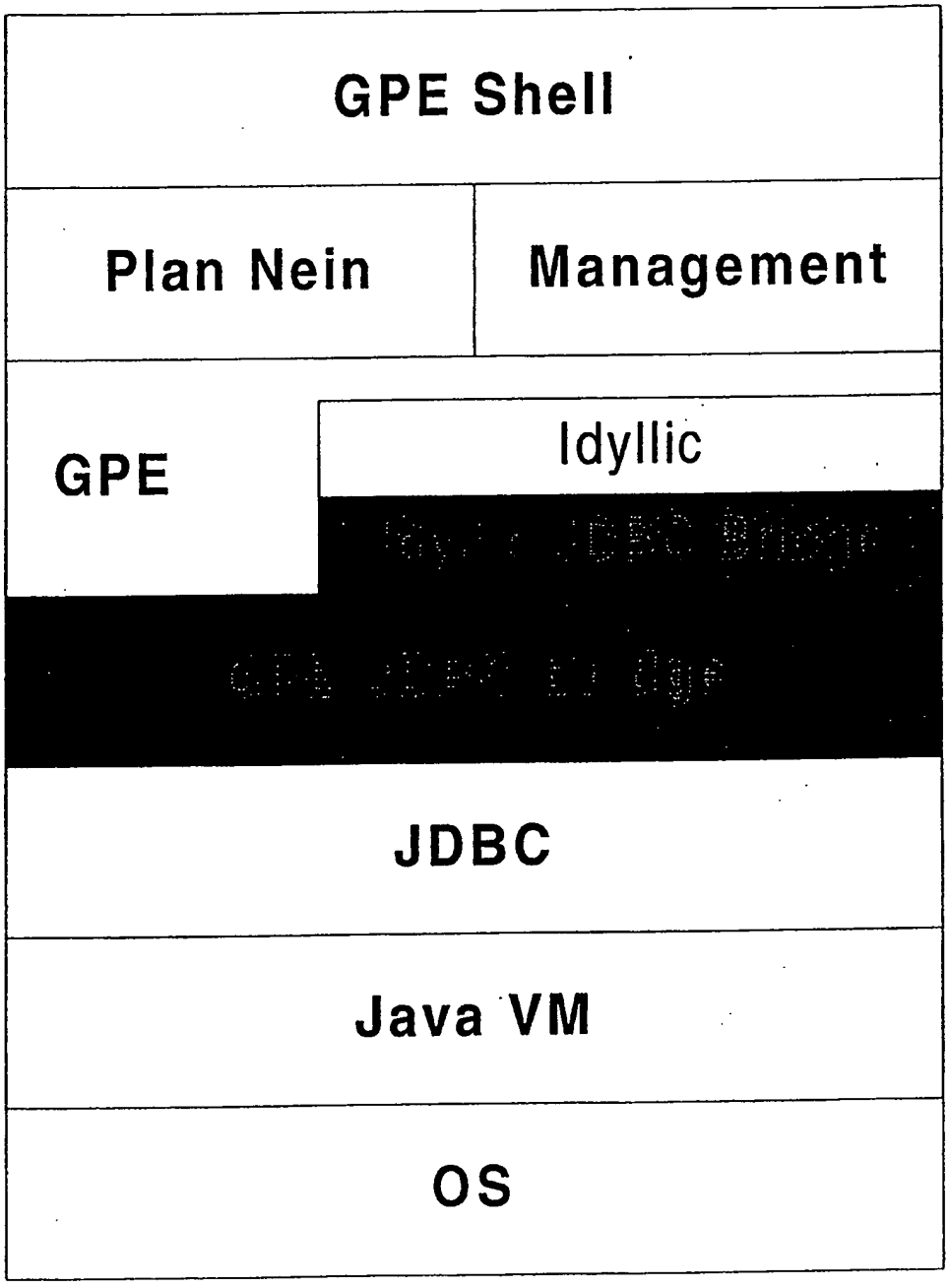


Fig. 5

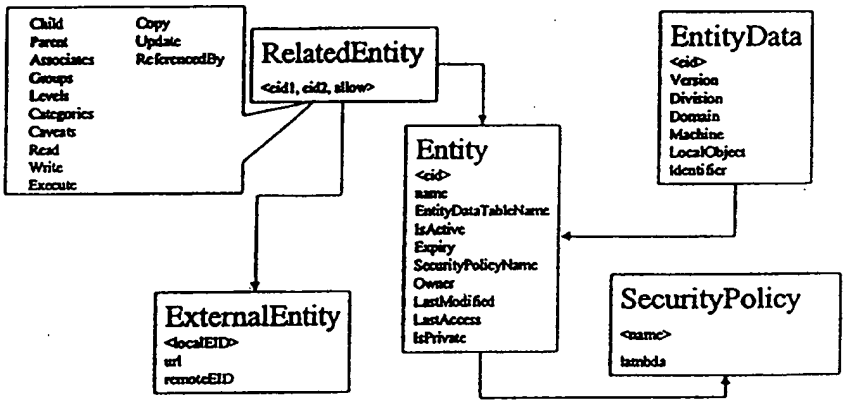

Child
Parent
Associates
Groups
Levels
Categories
Caveats
Read
Write
Execute
Copy
Update
ReferencedBy
RelatedEntity
<cid1, cid2, allow>
Entity
<cid>
name
EntityDataTableName
IsActive
Expiry
SecurityPolicyName
Owner
LastModified
LastAccess
IsPrivate
EntityData
<cid>
Version
Division
Domain
Machine
LocalObject
Identifier
ExternalEntity
<localEID>
url
remoteEID
SecurityPolicy
<name>
lambda

Fig.6

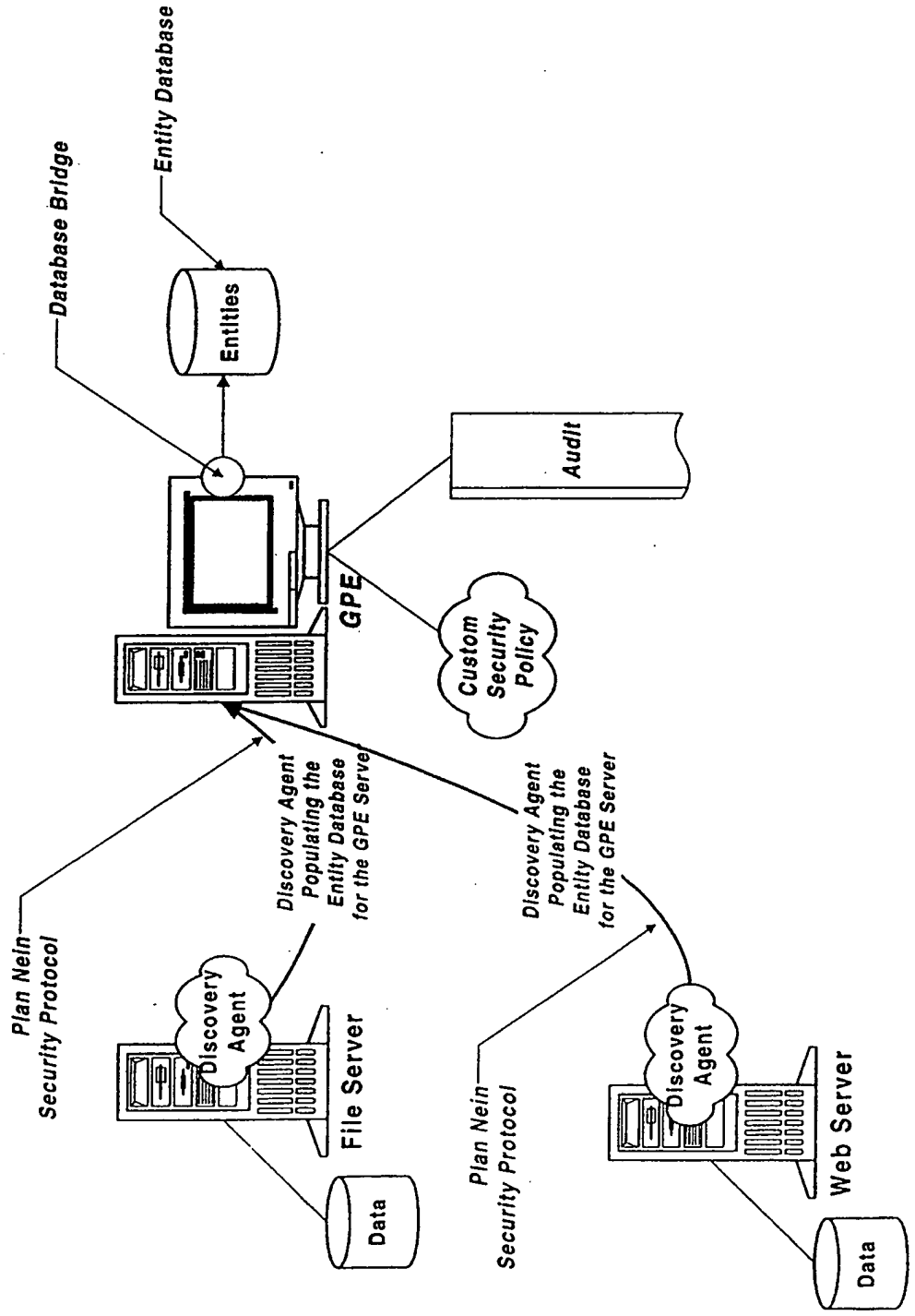

Entity Database
Database Bridge
Entities
Audit
GPE
Custom Security Policy
Plan Nein Security Protocol
Discovery Agent Populating the Entity Database for the GPE Server
Discovery Agent Populating the Entity Database for the GPE Server
Discovery Agent
File Server
Data
Plan Nein Security Protocol
Discovery Agent
Web Server
Data

Fig. 7

INTERNATIONAL SEARCH REPORT

In ational Application No
PCT/CA 00/00277

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L29/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	ZAHIR TARI: "Using agents for secure access to data in the Internet" IEEE COMMUNICATIONS MAGAZINE, 'Online! vol. 5, no. 6, - June 1997 (1997-06) pages 136-140, XP002142299 Royal Melbourne Institute, Australia ISSN: 0163-6804 Retrieved from the Internet: <URL:www.iel.ihs.com> 'retrieved on 2000-07-07! abstract page 136, right-hand column, line 13 - line 25 page 138, left-hand column, line 24 -right-hand column, line 26 page 138, right-hand column, line 39 - line 61 page 140, right-hand column, line 14 - line 37 -/--	1-7,9, 17-23,28

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
11 July 2000	25/07/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Adkhis, F

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/CA 00/00277

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 764 890 A (MCKELVIE SAMUEL J ET AL) 9 June 1998 (1998-06-09) abstract figure 1 column 1, line 7 - line 12 column 3, line 5 - line 46 column 4, line 43 -column 6, line 36	1-7,9, 17-23

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int ational Application No
PCT/CA 00/00277

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5764890 A	09-06-1998	NONE	